

# SQL Funkce

# SQL Funkce

- Funkce v SQL můžeme rozdělit na:
  - AgregáčnÍ – již jsme probírali
  - Skalární
    - číselné,
    - pro práci s řetězci,
    - konverznÍ,
    - pro práci s časem,
    - a další (např. práce s XML).
- Syntaxe funkcÍ je závislá na zvolené databázi. V této přednášce uvažujeme databázi Oracle. Nicméně i v jiných db je syntaxe zmíněných funkcÍ obdobná.

# Skalární funkce

- Skalární funkce se aplikují na jeden argument (parametr, sloupec) a vrací opět jednu hodnotu.
- Také se označují jako „jednořádkové“ funkce. Pracují s jedním řádkem (záznamem), přičemž pro každý řádek vracejí jednu hodnotu.
- Skalární funkce mohou pracovat s numerickými hodnotami, řetězci, datumovými a časovými hodnotami, mohou sloužit pro konverzi údajů atd.
- Skalární funkce, které budou zmíněné v přednášce, lze vyzkoušet zadáním syntaxe:

```
SELECT  nazev_funkce (...) FROM dual;
```

# Vnořování funkcí

- Funkce je možné vnořovat:

- Skalární funkce lze libovolně vnořovat:

```
select upper(trim(jmeno)) jmeno from zamestnanec;
```

- AgregáčnÍ funkce je možné vnořit max. dvakrát:

- Průměrný počet zaměstnanců na jedno oddělení:

```
select avg(count(zamestnanec_id)) from zamestnanec  
inner join zamestnani  
on zamestnani.zamestnani_id = zamestnanec.zamestnani_id  
group by zamestnani.zamestnani_id
```

- Je možné kombinovat agregáčnÍ i skalární funkce, například u předcházejícího SELECTu je možné změnit: avg(count(zamestnanec\_id)) za: trunc(avg(count(zamestnanec\_id)))

# Číselné funkce

- **ABS(N)** – vrací absolutní hodnotu čísla N  
Př.: **ABS** (-3.14) = 3.14
- **CEIL(N)** - vrací nejmenší celé číslo větší než N.  
Př.: **CEIL** (3.1415326) = 4
- **FLOOR(N)** – vrací největší celé číslo menší než N.  
Př.: **FLOOR** (3.1415326) = 3
- **MOD(M, N)** – zbytek po dělení M : N.  
Př.: **MOD** (8, 2.5) = 0.5
- **POWER(M, N)** – vrací hodnotu N-té mocniny čísla M.  
Př.: **POWER** (2, 3) = 8
- **SQRT(N)** – druhá odmocnina z N.

# Číselné funkce

- $\text{SIN}(N)$  – vrací hodnotu goniometrické funkce sinus,  $N$  je úhel v radiánech.  
Př.: **SIN**(1) = 0,84147
- $\text{COS}(N)$  – vrací hodnotu goniometrické funkce kosinus,  $N$  je úhel v radiánech.
- $\text{TAN}(N)$  – vrací hodnotu goniometrické funkce tangens,  $N$  je úhel v radiánech.
- $\text{ASIN}(N)$  – inverzní sinus, vrací hodnotu úhlu v radiánech.  
Př.: **ASIN**(**SIN**(1)) = 1
- $\text{ACOS}(N)$  – inverzní kosinus, vrací úhel v radiánech.
- $\text{ATAN}(N)$  – inverzní tangens, vrací úhel v radiánech.

# Číselné funkce

- **SIGN(N)** – zjišťuje, zda-li je číslo N záporné (vrací -1), kladné (vrací 1) nebo 0 (vrací 0).

Př.: **SIGN** (-0) = 0

- **ROUND(N [, M])** – zaokrouhlí číslo N na M desetinných míst. Pokud je M záporné, zaokrouhluje se na místa před desetinnou tečkou. Pokud není M zadáno, zaokrouhluje se na celé číslo.

Př.: **ROUND** (3.1415326, 3) = 3.142

- **TRUNC(N [, M])** – ořízne přebytečná desetinná místa. Funguje podobně jako funkce **ROUND(N [, M])**, zaokrouhluje však vždy směrem dolů.

Př.: **TRUNC** (3.1415326, 3) = 3.141

# Číselné funkce

- $\text{EXP}(N)$  – vrátí  $N$ -tou mocninu čísla  $e$ .  
Př.: **EXP** (1) = 2.71828
- $\text{LOG}(N, M)$  – vrátí hodnotu logaritmu o základu  $N$  z logaritmovaného čísla  $M$ .  
Př.: **LOG** (10, 1000) = 3
- $\text{LN}(N)$  – vrátí hodnotu přirozeného logaritmu z čísla  $N$ .  
Př.: **LN** (EXP (1) ) = 1
- $\text{BITAND}(N, M)$  – provede logický součin čísel  $N$  a  $M$ .  
Př.: **BITAND** (7, 9) = 1



# Funkce pro práci s řetězcí

- UPPER(X) – převede řetězec X na velká písmena.
  - Velice často se používá ve WHERE podmínce, když chcete získávat záznamy nezávisle na velikosti písmen dat v tabulce (také je možné použít LOWER):

```
select jmeno, prijmeni  
from zamestnanec  
where upper(jmeno) = upper('michael');
```
- LOWER(X) – převede řetězec X na malá písmena.
- INITCAP(X) – převede první písmeno v řetězci X na velké.

# Funkce pro práci s řetězcí

- **CHR(X)** – překonvertuje číselný kód X na znak.  
Př.: **CHR**(65) = 'A'
- **ASCII(X)** – překonvertuje znak X na číselný kód.  
Př.: **ASCII**('A') = 65
- **SUBSTR(řetězec, začátek [, délka])** – vrací podmnožinu vstupního řetězce, která je definovaná začátkem a počtem znaků (délkou).  
Př.: **SUBSTR**('Autobus', 5, 3) = 'bus'

# Funkce pro práci s řetězcí

- **REPLACE**(řetězec1, řetězec2 [, řetězec3]) – vrací nový řetězec, vytvořený z původního řetězce1, přičemž nahradí všechny výskyty řetězce2 řetězcem3. Pokud parametr řetězec3 chybí, jsou všechny výskyty řetězce2 v řetězci1 vymazány.

Př.: **REPLACE**('Jack and Jue', 'J') = 'ack and ue'

Př.: **REPLACE**('Jack and Jue', 'J', 'Bl') =  
'Black and Blue'

- **TRANSLATE**(řetězec1, řetězec2, řetězec3) – nahradí v řetězci1 znaky z řetězce2 odpovídajícími znaky z řetězce3. Ostatní znaky jsou ponechány.

Př.: **TRANSLATE**('abceda', 'e', '-') = 'ab-c-da'

# Funkce pro práci s řetězcí

- **TRIM**([**LEADING**|**TRAILING**|**BOTH**] [**x FROM**] řetězec)  
- odstraní znak x z patričného konce řetězce. Pokud není zadán konec řetězce, dosadí se **BOTH**. Pokud není zadán znak x, dosadí se mezera.

Př.: **TRIM**('     Ab   ') = 'Ab'

Př.: **TRIM**(**LEADING FROM** '   Ab   ') = 'Ab   '

Př.: **TRIM**('0' **FROM** '00Ab000') = 'Ab'

Př.: **TRIM**(**LEADING** '0' **FROM** '00Ab000') = 'Ab000'

- **LTRIM**(řetězec1 [, řetězec2]) – ořezání řetězce1 o znaky z řetězce2 zleva. Při nezadání řetězce2 je dosazena mezera.

Př.: **LTRIM**('aabbccAaBab', 'abc') = 'AaBab'

- **RTRIM**(řetězec1 [, řetězec2]) – ořezání řetězce1 o znaky z řetězce2 zprava.

# Funkce pro práci s řetězci

- LPAD(řetězec1, délka [, řetězec2]) – doplní řetězec1 na požadovanou délku znaky z řetězce2 zleva. Při nezadání řetězce2 se bere mezera.

Př.: **LPAD** ( 'Rybka' , 8 , '<>' ) = '<><Rybka'

- RPAD(řetězec1, délka [, řetězec2]) – doplní řetězec1 na požadovanou délku znaky z řetězce2 zprava.

Př.: **RPAD** ( 'Rybka' , 8 , '<>' ) = 'Rybka<><'

- CONCAT(řetězec1, řetězec2) – vrací spojení řetězce1 a řetězce2. Obdoba řetězec1 || řetězec2.

Př.: **CONCAT** ( 'Auto' , 'bus' ) = 'Autobus'

- LENGTH(řetězec) – vrací délku řetězce.

Př.: **LENGTH** ( 'Auto' ) = 4

# Funkce pro práci s řetězcí

- **INSTR**(řetězec1, řetězec2 [, N [, M]]) – od N-tého znaku hledá v řetězci1 M-tý výskyt řetězce2. Pokud je N záporné, hledá v opačném směru. Vráť pozici nalezeného výskytu, nebo 0.

Př.: **INSTR**('Autobusu', 'u') = 2

Př.: **INSTR**('Autobusu', 'u', 3) = 6

Př.: **INSTR**('Autobusu', 'u', 3, 2) = 8

# Konverzní funkce

- TO\_CHAR(datum [, format]) - převede datum na řetězec podle zadaného formátu.

Př.: **TO\_CHAR** ( **sysdate**, 'MM-DD-YYYY' )

- TO\_CHAR(cislo [, format]) - převede číslo na řetězec, aby byly zachovány všechny platné číslice.

Př.: **TO\_CHAR** (1, '9.999') = '1.000'

- TO\_DATE(řetězec [,format]) - převede řetězec na datum dle zadaného formátu.

Př.: **TO\_DATE** ( '22.02.09' , 'DD.MM.YY' )

- TO\_NUMBER(řetězec [, format]) - převede řetězec na číslo.

Př.: **TO\_NUMBER** ( '1.000' , '9.999' )

# Konverzní funkce

- Všechny možné hodnoty pro formát v konverzních datumových funkcích lze nalézt v dokumentaci. Přehledněji např. na stránkách:
  - [http://www.techonthenet.com/oracle/to\\_date.php](http://www.techonthenet.com/oracle/to_date.php)
- Nejčastěji používané:

Zkratka	Popis
DD	Den v měsíci (1 – 31)
MM	Měsíc (01 – 12)
YYYY	Rok, např. 2013
HH24	hodina dne (0 – 23)
MI	Minuta (0 – 59)
SS	Sekunda (0 – 59)

- Všechny možné hodnoty pro formát v konverzních číselných funkcích lze nalézt v dokumentaci, ale nejčastěji se používá devítka a tečka jak je uvedeno na přecházejícím snímku.



# SYS\_CONTEXT

- SYS\_CONTEXT(n, p [, délka]) – slouží k získávání systémových informací, n = namespace, p = parametr, asociovaný s namespace.
- Př.: Získá IP adresu, ze které se připojuje uživatel:  

```
select sys_context('userenv', 'ip_address') from dual;
```
- Př.: Získá formát data pro aktuální session:  

```
select sys_context('userenv', 'nls_date_format') from dual;
```
- Plný seznam parametrů v namespace USERENV:
  - [http://docs.oracle.com/cd/E14072\\_01/server.112/e10592/functions182.htm](http://docs.oracle.com/cd/E14072_01/server.112/e10592/functions182.htm)

# NLS\_DATE\_FORMAT I.

- Oracle provádí automatickou konverzi řetězce na datum:
  - Výběr všech zaměstnanců, kteří nastoupili do aerolinek po 1.1.2009:

```
SELECT jmeno, prijmeni, datum_nastupu  
FROM zamestnanec  
WHERE datum_nastupu > '01.01.09';
```

- V jakém formátu bude tento řetězec závisí na hodnotě proměnné NLS\_DATE\_FORMAT, viz. následující snímek.

# NLS\_DATE\_FORMAT II.

- Získání aktuální hodnoty NLS\_DATE\_FORMAT:

```
select sys_context('userenv', 'nls_date_format') from dual;
```

- Změna NLS\_DATE\_FORMAT pro aktuální session:

```
ALTER SESSION SET nls_date_format = 'DD-Mon-YYYY';
```

- Poté pro aktuální session předcházející způsob implicitní konverze datumu nebude fungovat, místo toho je nutné použít takový SELECT:

```
SELECT jmeno,  
       prijmeni,  
       datum_nastupu  
FROM   zamestnanec  
WHERE  datum_nastupu > '01.Led.09';
```

# Funkce pro práci s časem

- SYSDATE – vrací aktuální datum a čas.
- CURRENT\_DATE – vrací aktuální datum a čas, ale s nastavením časového pásma.
- LOCALTIMESTAMP – vrací aktuální čas.
- CURRENT\_TIMESTAMP – vrací aktuální čas, ale s nastavením časového pásma.
- K vráceným hodnotám lze přičítat čísla, kde 1 je jeden den  
=> 0.25 je 6 hodin, 1/24 je jedna hodina apod.

# Funkce pro práci s časem

- **LAST\_DAY(datum)** – vrací poslední den v měsíci zadaného datumu.

Př.: **LAST\_DAY** ('22.2.09') = '28.2.09'

Př.: **LAST\_DAY(sysdate) - sysdate** ... vrací počet dní do konce měsíce

- **NEXT\_DAY(datum, den)** – vrátí datum dne příští týden.

Př.: **NEXT\_DAY** ('03.09.09', 4) = '10.09.09'

... 03.09.09 je čtvrtek, další čtvrtek (4.den) je '10.09.09'

# Funkce pro práci s časem

- **ADD\_MONTHS(datum, N)** – k zadanému datumu přičte N měsíců.

Př. **ADD\_MONTHS** ('03.09.09', 4) = '03.01.10'

- **MONTHS\_BETWEEN(datum1, datum2)** – vrátí počet měsíců mezi uvedenými daty (datum1 - datum2).

Př. **MONTHS\_BETWEEN** ('03.09.09', '03.01.10') = -4

- **EXTRACT (**  
  { YEAR | MONTH | DAY | HOUR | MINUTE | SECOND }  
  | { TIMEZONE\_HOUR | TIMEZONE\_MINUTE }  
  | { TIMEZONE\_REGION | TIMEZONE\_ABBR }  
  **FROM** { datum | interval } )
- vytažení hodnoty z datumu nebo intervalu.

Př.: **EXTRACT (YEAR FROM sysdate)**

# Funkce pro práci s časem

- `NEW_TIME(date, tz1, tz2)` – vrátí datum v časové zóně `tz2` pro datum „`date`“, které se nachází v časové zóně `tz1`.
- Bohužel `NEW_TIME` podporuje pouze pár Amerických časových zón a GMT, plný seznam:
  - [http://docs.oracle.com/cd/B19306\\_01/server.102/b14200/functions092.htm](http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions092.htm)

# Funkce pro práci s časem

- **ROUND(datum [, přesnost])** – zakoukrouhlí datum s uvedenou přesností. Přesnost může být: YEAR, MONTH, DD (celé dny), DAY (první den v týdnu), HH (hodiny), MI (minuty) **a další**. Pokud není přesnost zadána berou se dny.  
Př. `ROUND (TO_DATE ( '22.02.09 12.02' ,  
                              ' DD.MM.YY HH.MI ' ) ) = '23.02.09'`
- **TRUNC(datum, [, přesnost])** – ořízne datum s uvedenou přesností. Funguje podobně jako funkce **ROUND(datum, [, přesnost])**, zaokrouhluje však vždy směrem dolů.



# Funkce pro práci s časem

- Při práci s časem můžete také použít datový typ INTERVAL, který obsahuje nějaký časový interval. K čemu se to může hodit? Například uděláte rozdíl mezi dvěma daty:

```
select to_date('24.12.2015') - sysdate from dual;
```

- A vyjde vám například něco takového: 23,6868402777778 (což znamená, že do Vánoc zbývá 23 dní a nějaké drobné ... otázka je, co ty „drobné“ jsou). Tímto způsobem to dokážete zjistit:

```
select numtodsinterval(0.6868402777778, 'day') from dual;
```

- Funkce NUMTODSINTERVAL konvertuje číslo na INTERVAL, který obsahuje hodiny, minuty, sekundy, ...

# Funkce pro práci s časem

- Získání jednotlivých částí intervalu:

```
with muj_interval as
  (select numtodsinterval(0.12345, 'day') hodnota from dual)
select hodnota,
       extract(hour from hodnota),
       extract(minute from hodnota),
       extract(second from hodnota)
from muj_interval;
```

# Další SQL funkce

- SOUNDEX(parametr) – tato funkce vrátí Soundex fonetickou reprezentaci parametru. Tato funkce se obvykle používá k nalezení jmen, která zní podobně.

```
select jmeno, prijmeni from pasazer  
where soundex(jmeno) = soundex('Radek')
```

# Další SQL funkce

- GREATEST(expr\_list) – vstupem je jeden argument, který je listem výrazů. Tato funkce vrátí výraz, který je největší. Jestli se porovnává datum, text nebo číslo je závislé na prvním výrazu v listu.
  - Příklad: `SELECT GREATEST('1.1.2009', '1.1.2010') FROM dual;`
- **Pozor! Neplést si GREATEST() s MAX()!**
  - GREATEST je skalární funkce, která pro každý řádek vrací jednu hodnotu.
  - MAX je agregační funkce, která pro sloupec vrací jednu hodnotu.
- LEAST(expr\_list) – obdobné jako GREATEST, ale vrací výraz, který je nejmenší.
- **Obdobně neplést si LEAST() s MIN()!**

# Další SQL funkce

- NVL(paramert1, paramert2) - vrací paramert2, pokud je paramert1 null. Jinak vrátí paramert1.

Př. **NVL**(null, 4) = 4

- NVL2(paramert1, paramert2, paramert3) – rozšíření funkce NVL. Vrací paramert3, pokud je parametr1 null. Jinak vrátí paramert2. Používá se, pokud je potřeba ošetřit jak hodnotu null, tak samotnou hodnotu v parametru1.

Př.: **NVL2**('Nová', 'Nová' || ' ulice', 'Bez ulice')

Pozn.: lepší použití je s proměnnou (sloupcem) jako parametrem1.

# Další SQL funkce

- `NULLIF(x1, x2)` – tato funkce vrátí `NULL`, když  $x1 = x2$ , v opačném případě vrátí `x1`. Když `x1` je `NULL`, pak `NULLIF` vrátí `NULL`.

# Další SQL funkce

- COALESCE je zobecnění NVL funkce. Do funkce COALESCE vstupuje list parametrů oddělených čárkou. Tato funkce vrátí první nenullovou hodnotu z těchto parametrů.
- Například COALESCE (x1, x2, x3) se bude vyhodnocovat následovně:
  - Když je x1 null, pokračuj na x2, jinak vrať x1 a ukonči běh funkce.
  - Když je x2 null, pokračuj na x3, jinak vrať x2 a ukonči běh funkce.
  - Když je x3 null, vrať null, jinak vrať x3.

# Další SQL funkce

- `DECODE(výraz, hledaný_výraz1, výsledek1  
[, hledaný_výraz2, výsledek2 ]...  
[, defaultní_výsledek ]  
)`
  - funkce postupně porovnává hodnotu výrazu s hodnotami hledaných výrazů. Pokud se hodnota výrazu rovná hodnotě hledaného výrazu, vrátí uvedený výsledek. Pokud se hodnoty nikde neshodují, vrátí defaultní výsledek (pokud není uveden, vrátí null). Při porovnávání hodnoty výrazu s hodnotou hledaného výrazu jsou považovány hodnoty null za shodné.
- `Př.: SELECT jmeno, prijmeni,  
DECODE (problematicky, 'y', 'ano', 'ne' ) problem_pasazer  
FROM pasazer;`



# CASE I.

- CASE – není to funkce, ale příkaz, který je čitelnější než DECODE funkce a umí mnohem víc. Př.:

```
SELECT jmeno, prijmeni,  
       CASE problematicky  
         WHEN 'y'  
         THEN 'ano'  
         ELSE 'ne'  
       END problematicky_pasazer  
FROM pasazer;
```

# CASE II.

- Maximální počet argumentů v CASE výrazu je 255.
- CASE také umožňuje vrátit hodnoty na základě podmínky:

```
SELECT jmeno, prijmeni, plat,  
CASE  
    WHEN plat < 100000  
    THEN 'maly'  
    WHEN plat BETWEEN 100000 AND 200000  
    THEN 'standard'  
    ELSE 'nadstandard'  
END velikost_platu  
FROM zamestnanec;
```

# ROWID I.

- Každý záznam v databázi obsahuje unikátní hodnotu, která se nazývá ROWID.
- ROWID má následující použití:
  - Jedná se o nejrychlejší způsob zpřístupnění jednoho řádku.
  - Je to unikátní identifikátor pro řádky v tabulce.
- Získání ROWID:

```
SELECT rowid, jmeno, prijmeni
```

```
FROM ZAMESTNANEC
```

```
WHERE PRIJMENI = 'Boss';
```

- Výsledek (ROWID se s největší pravděpodobností bude oproti tomuto výsledku lišit):

ROWID	JMENO	PRIJMENI
AAAE5XAABAAAK+hAAA	Michael	Boss

# ROWID II.

- Můžete také použít funkci ROWIDTOCHAR, která provede explicitní konverzi rowid na text:

```
SELECT ROWIDTOCHAR(rowid), jmeno, prijmeni  
FROM ZAMESTNANEC  
WHERE PRIJMENI = 'Boss';
```

- Zpřístupnit záznam pomocí ROWID můžete následovně:

```
SELECT * FROM ZAMESTNANEC  
WHERE ROWID = 'AAAE5XAABAAAK+hAAA';
```

- Můžete také použít funkci CHARTOROWID, která provede explicitní konverzi textu na rowid:

```
SELECT * FROM ZAMESTNANEC  
WHERE ROWID = CHARTOROWID('AAAE5XAABAAAK+hAAA');
```

# ROWID III.

- V Toadu (a PL/SQL Developeru) můžete přímo ve výsledku SELECTu měnit data v databázi když v SELECTu uvedete rowid (což je speciální sloupec, který je v každé tabulce a slouží pro nejrychlejší přístup k řádku). Když ho v SELECTu uvedete, pak pomocí něj Toad mění záznamy v příslušném řádku:

The screenshot shows the Toad SQL Developer interface. At the top, a SQL query is entered in the editor:

```
1 ► select rowid, e.* from employees e
```

Below the editor is the 'Data Grid' tab, which displays the results of the query. The grid has columns: EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE\_NUMBER, and HIRE\_DATE. The first row is highlighted, showing the rowid '1' and the employee details for Steven King.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
I	100	Steven	King	SKING	515.123.4567	17/6/2003
	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/9/2005
	102	Lex	De Haan	LDEHAAN	515.123.4569	13/1/2001

At the bottom of the interface, the status bar shows '1: 35', '13 msec', 'Row 1 of 107 total rows', and the connection string 'HR@LOCALHOST:1521/HR'.

# ORA\_HASH

- ORA\_HASH je funkce, která slouží pro získávání náhodného výběru záznamů z tabulky.
- Má následující syntaxi: ORA\_HASH(expr [, max\_bucket [, seed]])
  - expr = výraz, pro který se počítat hash
  - max\_bucket = počet generovaných skupin
  - seed = číslo (větší než 0), slouží k generování různých skupin
- Rozdělení zaměstnanců do deseti skupin pomocí funkce ORA\_HASH:
  - **SELECT ORA\_HASH(ZAMESTNANEC\_ID, 10), JMENO, PRIJMENI  
FROM ZAMESTNANEC;**
- Náhodný výběr zaměstnanců z nulté skupiny:
  - **SELECT JMENO, PRIJMENI FROM ZAMESTNANEC WHERE  
ORA\_HASH(ZAMESTNANEC\_ID, 10) = 0;**